Genetic Algorithms (GAs)

- heuristic methods—do not guarantee optimality
- iterative methods
- simulate biological evolution, which has enabled "nature" to develop species remarkably well-adapted to their environment.

GAs

- operate on a population of individuals which represent potential solutions to the problem
- from the population, a set of "good" individuals are selected to mate & form a new generation
- the quality of subsequent generations will (it is hoped!) gradually improve and approach optimality

Encoding

Each individual in the population consists of a string, which "resembles" a biological chromosome composed of genes.

These genes may take on values from a finite set of digits or characters, called alleles. (Most frequently, the alleles are binary digits.)

Example: from the set of alleles {A, B, C} we might produce the individual ABCABBACBB

Fitness

For each individual \(i\), we define a fitness value \(f(i)\) to measure how good it is.

Individuals representing good solutions will have a larger fitness value than those which correspond to poor solutions.

Used to determine which individuals survive to produce the next generation.

"Survival of the fittest..."
From the population at each generation, a GA randomly selects the fittest individuals to survive and mate to produce the next generation.

The simplest method for doing this is "stochastic sampling with replacement", in which individual \( i \) is selected with probability

\[
p_i = \frac{f(i)}{\sum_{j=1}^{N} f(j)}
\]

This is repeated until \( N \) individuals have been selected.

In sampling with replacement, the more fit individuals are likely to be selected, and may be selected several times. However, there is no guarantee that an individual whose fitness is above the population average will be selected.

"Stochastic universal sampling" is a scheme which will guarantee this!


1. Compute \( S \), the sum of the fitness values for the population.

2. Map each fitness value in random order to contiguous segments of the interval \([0, S] \) on the real line, such that each segment has length equal to its corresponding fitness value.

3. Let \( n_k \) be the number of values in the set \( \{s_i\} \) which fall within interval \( k \).

4. Compute the \( N \) equally-spaced numbers

\[
s_i = \frac{s_0 + i \times \frac{S}{N}}{N}, \quad i = 0, 1, 2, \ldots, N-1
\]

5. For each \( k \) (\( k = 1, 2, \ldots, N \)), create \( n_k \) copies of individual \( k \) and put into the mating pool.

A way to prevent this is to scale the fitness values so as to maintain a constant "best fitness to average fitness" ratio, the scaling factor.

Given scaling factor \( \lambda \) and a set of fitness values \( \{f_1, f_2, \ldots, f_N\} \), we scale them to \( F_i = a f_i + b \) where \( a \) & \( b \) are chosen so that

\[
\sum_{i=1}^{N} F_i = 1
\]

and

\[
\lambda = \frac{\max F_i}{\sum_{i=1}^{N} F_i / N}
\]

Solution: Choose \( a \) & \( b \) so that

\[
a = \frac{\lambda - 1}{N f_{\text{max}} - \sum_{i=1}^{N} f_i}
\]

\[
b = \lambda f_{\text{max}} - a f_{\text{max}}
\]

where \( f_{\text{max}} = \max f_i \)
Two individuals can mate to generate two offspring, by randomly selecting a break point and exchanging substrings:

\[ \text{ABCAABCC} \rightarrow \text{ABCAACBBCC} \rightarrow \text{ABCAABAACBCC} \]

**CROSSOVER**

The GA is initiated with a random population of \( N \) individuals.

At each iteration, the GA does the following:

1. Evaluates the fitness of each individual in the population.
2. Based on fitness, chooses individuals from the population based on fitness, to form a mating pool, and pairs these individuals randomly.

There are numerous decisions which much be made in the design of a genetic algorithm:

- population size
- probability of crossover
- probability of mutation
- whether to "seed" the initial population with "good" solutions
- scaling factor (best to average ratio) for fitness


**MUTATION**

With relatively low probability, a gene on a chromosome might change its value from one allele to another.

\[ \text{ABCAABCC} \rightarrow \text{ABCAABBCC} \rightarrow \text{ACCAABBCC} \]

**GENETIC ALGORITHM FOR LINE BALANCING**

Consider the Assembly Line Balancing (ALB) Problem in which the number of stations \( n \) is fixed, and the tasks are to be assigned to the stations so as to minimize the cycle time.

**EXAMPLE**

\( n = 3 \) stations
10 tasks

**ENCODING**

For the ALB problem, a natural encoding would be, using the station numbers as alleles, to identify the station to which task \( i \) is assigned by the number in the \( i \)th position on the string.
The assignment shown above would be coded by the string: \[1121223233\]

The cycle time is to be minimized, and so is an indication of "unfitness", i.e., the larger the cycle time, the more unfit the solution is.

If \(T(i)\) is the cycle time of individual \(i\), then the fitness could be measured by

\[f(i) = k \times M - T(i)\]

where \(M = \max |T(i)|\)

and \(k > 1\). With this definition, \(f(i) > 0\) for all \(i\).

However, because individuals which violate one or more precedence restrictions might appear in the population, we will include a penalty \(P\) times the number \(V\) of such violations:

\[T(i) = T_{\text{max}} + P \times V\]

In the assignment shown, task 6 does not precede task 8 as required, and so we reduce the fitness by a penalty \(P = 10\):

\[T(i) = 43 + 10 \times 1 = 53\]
Genetic Algorithm for Line Balancing

Consider the ALB problem used as an example earlier.

Genetic Algorithm Parameters:
- 08:21:59 on December 2, 1995
- Population size: 60
- Fittest = 0.8
- Mutation rate = 0.009
- Penalty = 5
- N = penalty per precedence violation

At the 100th iteration, the algorithm converged:

The incumbent was discovered in an earlier generation, but then disappeared.

Running again, with a higher probability of crossover (90%),
Again, the algorithm converged (population is not uniform, but has uniform fitness).

Mutation will be performed on each "gene" with some probability \( P_{\text{mut}} \), which is relatively small.

If a gene is randomly selected for mutation, it is assigned an allele selected uniformly from the set of alleles.

The incumbent: [1 2 3 4 5 6 7 8 9] ∈ [1 2 3 4 5 6 7 8 9]
Genetic Algorithm for Line Balancing

Cycle time is 45

Running the GA again with P<sub>cross</sub>=90%, it converged in 88th generation:

<table>
<thead>
<tr>
<th>generation</th>
<th>Fitness</th>
<th>min</th>
<th>max</th>
<th>average</th>
<th>#feasible</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>91</td>
<td>110</td>
<td>101.96</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>73</td>
<td>92</td>
<td>111</td>
<td>101.15</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>74</td>
<td>96</td>
<td>111</td>
<td>102.80</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>75</td>
<td>92</td>
<td>111</td>
<td>104.76</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>88</td>
<td>111</td>
<td>105.70</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>77</td>
<td>92</td>
<td>111</td>
<td>104.99</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>78</td>
<td>92</td>
<td>111</td>
<td>105.49</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>79</td>
<td>89</td>
<td>111</td>
<td>105.95</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>99</td>
<td>111</td>
<td>107.70</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>81</td>
<td>101</td>
<td>111</td>
<td>106.70</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>82</td>
<td>102</td>
<td>111</td>
<td>107.70</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>83</td>
<td>102</td>
<td>111</td>
<td>107.70</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>84</td>
<td>102</td>
<td>111</td>
<td>107.70</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>85</td>
<td>102</td>
<td>111</td>
<td>107.70</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>86</td>
<td>102</td>
<td>111</td>
<td>107.70</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>87</td>
<td>102</td>
<td>111</td>
<td>107.70</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>88</td>
<td>102</td>
<td>111</td>
<td>107.70</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Converged at iteration 104

Converged to non-feasible solution!

The incumbent was discovered in an earlier generation, and then disappeared!
Running the GA again, but with population size = 30, not 20

Converges in 76th generation
Finally, the GA was run with a population size of 50, and a probability of crossover equal to 75%. Convergence occurred on the 77th iteration.
All solutions in the final population are feasible, with fitness = 167 and cycle time = 43.

There are four distinct solutions in the population:

- 1 2 3 4 5 6 7 8 9
- 1 2 3 4 5 6 7 8 9
- 1 2 3 4 5 6 7 8 9
- 1 2 3 4 5 6 7 8 9

The best solution was found in the 35th generation, with a cycle time equal to 40.

The optimal cycle time is 39.

The following screens contain listings of some APL functions written to implement GA for the Assembly Line Balancing problem.
Genetic Algorithm for Line Balancing

A genetic algorithm is used to solve the line balancing problem. The algorithm is designed to find the optimal assignment of tasks to stations on a production line. The fitness of each individual is evaluated based on the total time and the number of tasks assigned to each station. The algorithm employs selection, crossover, and mutation operators to evolve the population towards an optimal solution.

The program consists of several procedures, including:
- Initial_Pool: Generates an initial population for the genetic algorithm.
- Select_Pool: Selects a pool of individuals who will survive to the next generation.
- Report_Status: Reports the status of the population at each generation.
- Evaluate_Pool: Evaluates the fitness of the population.
- Compute_Cycle_T: Computes the cycle time for the solution.
- Next, Next++, NextEnd, NextEnd, NextEnd: Control flow statements for the algorithm.

The algorithm iterates until a convergence criterion is met, such as a maximum number of iterations or a fitness threshold.

Please refer to the code snippet for a more detailed understanding of the algorithm.
/* Genetic Algorithm for Line Balancing 

@Select_Pool P1;P2;P3;P4;P5;P6;P7;P8;P9;P10;P11;P12;P13;P14;P15;P16;P17;P18;P19

@ Select individuals from pool according to their fitness values
@ Fitness values are scaled so as to sum to 1.0 and
@ Max_Fitness = scale_factor - 1
@ Global variables (OA_Scale_Factor)
@ ar<(OA_Scale_Factor-1)/((Popsize*x/F)+1/F)
@ bx<-ax/F)
@ P=1/(x/Popsize+popsize)*ax/F
@ P=1
@ S=x/F or f=x/F
@ s0=(8-n)/8*0.001*x^2.000
@ ax=0.5*(8-N)/((N-1)*N)
@ n=0.5*(1+n)/(f+x)
@ 0 if Popsize=p-{((n>0)\(n\(0)\(n\(1)\)N

/* Genetic Algorithm for Line Balancing 

@ Select individuals from the population, who will survive to the next generation.
@ Arguments: Fitness = vector of fitness values of the individuals in the population.
@ Result: Q = vector of number of copies of individuals to be included in the next gen
@ Method used is Stochastic universal sampling due to J.E. Baker
@ Q=+/Fitness * N+Fitness * F+/Fitness
@ s0=(71000)+100000
@ ax=(x^2)/N+10000
@ s0=x+S*(N)=0,N
@ Q(+/F,1)=N/(+/Q,14F,1)+Q

--Generic Broiler, U of Iowa, 1997